On Faster Marginalization with Squared Circuits via Orthonormalization

Lorenzo Loconte[1] **Antonio Vergari**[1]

[1] School of Informatics, University of Edinburgh, UK

l.loconte@sms.ed.ac.uk, avergari@ed.ac.uk

Abstract

Squared tensor networks (TNs) and their generalization as parameterized computational graphs – *squared circuits* – have been recently used as expressive distribution estimators in high dimensions. However, the squaring operation introduces additional complexity when marginalizing variables or computing the partition function, which hinders their usage in machine learning applications. Canonical forms of popular TNs are parameterized via unitary matrices as to simplify the computation of particular marginals, but cannot be mapped to general circuits since these might not correspond to a known TN. Inspired by TN canonical forms, we show how to parameterize squared circuits to ensure they encode already normalized distributions. We then use this parameterization to devise an algorithm to compute any marginal of squared circuits that is more efficient than a previously known one. We conclude by formally showing the proposed parameterization comes with no expressiveness loss for many circuit classes.

1 Introduction

Tensor networks (TNs) are low-rank tensor factorizations often used to compactly represent high-dimensional probability distributions, both in quantum physics (Orús 2013; Biamonte and Bergholm 2017) and in ML (Stoudenmire and Schwab 2016; Glasser, Pancotti, and Cirac 2018; Cheng et al. 2019; Glasser et al. 2019; Novikov, Panov, and Oseledets 2021). A TN factorizing a complex function ψ over a set of variables $\mathbf{X} = \{X_i\}_{i=1}^d$ having domain $\mathsf{dom}(\mathbf{X})$ can be used to represent a probability distribution via modulus squaring, i.e., $p(\mathbf{X}) = Z^{-1}|\psi(\mathbf{X})|^2 = Z^{-1}\psi(\mathbf{X})\psi(\mathbf{X})^*$, where $(\cdot)^*$ denotes the complex conjugation, and $Z = \int_{\mathsf{dom}(\mathbf{X})} |\psi(\mathbf{x})|^2 \mathrm{d}\mathbf{x}$ is the partition function.

Recently, Loconte et al. (2023, 2024b) showed that TNs can be generalized into deep computational graphs called *circuits* (Choi, Vergari, and Van den Broeck 2020). This is done by casting tensor contraction operations into layers of sums and products whose feed-forward evaluation corresponds to a complete contraction to evaluate ψ. The language of circuits offers the opportunity to flexibly build novel TN structures by just stacking layers of sums and products as "Lego blocks" (Loconte et al. 2024b); include different basis input functions, and offering a seamless integration with deep learning (Shao et al. 2022; Gala et al. 2024a,b). Furthermore, casting TNs as circuits provides conditions as to compose them and compute many probabilistic reasoning tasks in closed-form, such as expectations and information-theoretic measures (Vergari et al. 2021), which is crucial, e.g., for reliable neuro-symbolic AI (Ahmed et al. 2022; Zhang et al. 2023). This is done with *probabilistic circuits* (PCs), circuits encoding probability distributions, that are classically restricted to have positive parameters only – also called *monotonic* PCs (Shpilka and Yehudayoff 2010).

One can increase the expressiveness of PCs by equipping them with complex parameters, *squaring* them (Loconte et al. 2024a), similarly to TNs, or even more by mixing squared PCs (Loconte, Mengel, and Vergari 2024). Differently from classical monotonic PCs, which are not squared, the squared PCs require additional computation to be normalized, i.e., to compute Z, which is quadratic in the circuit size. This computational overhead carries over to computing marginals and hinders their application in a number of tasks such as lossless compression (Liu, Mandt, and Van den Broeck 2022) and sampling (Loconte et al. 2024a), where performing fast marginalization is crucial.

In this paper, we show that the solution to this inefficiency of squared PCs comes from the literature of TNs, where *canonical forms* are adopted in order to simplify the computation of probabilities (Schollwoeck 2010). For instance, instead of computing Z explicitly in the case of a matrix-product state (MPS) TN (Pérez-García et al. 2007), a canonical form ensures $|\psi(\mathbf{X})|^2$ is an already-normalized probability distribution, i.e., $Z = 1$. In practice, canonical forms are obtained by parameterizing a TN using unitary matrices, i.e., matrices $\mathbf{A} \in \mathbb{C}^{n \times n}$ satisfying $\mathbf{A}^\dagger\mathbf{A} = \mathbf{A}\mathbf{A}^\dagger = \mathbf{I}_n$, where $\mathbf{I}_n$ denotes the identity matrix of size n. For rectangular matrices, *semi-unitary* matrices $\mathbf{A} \in \mathbb{C}^{m \times n}$ are considered, i.e., matrices satisfying $\mathbf{A}^\dagger\mathbf{A} = \mathbf{I}_n$ if $m > n$ or $\mathbf{A}\mathbf{A}^\dagger = \mathbf{I}_m$ if $m < n$. Under this view, computing Z simplifies into operations over identity matrices. However, computing different marginals over different TN structures requires a different canonical form and these cannot be immediately translated to squared PCs, because the latter allows us to build factorizations that might not correspond to any known TN. This begs the question: *how can we parameterize squared circuits as to be already normalized and allow*

us to accelerate the computation of any *marginal*? We answer it in the following.

Contributions. We derive sufficient conditions via orthonormal parameterizations to ensure that squared PCs are already normalized (Section 3). These conditions are based on semi-unitary matrices, similarly to canonical forms in TNs, but defined within the language of tensorized circuits instead. Then, by leveraging squared orthonormal PCs, we present a general algorithm to compute any marginal that can be much more efficient than the previously known algorithm for squared PCs which required a quadratic increase in complexity instead (Section 4). Our algorithm, which exploits the concept of variable dependencies of the circuit layers, can be used to speed up the computation or arbitrary marginals for TNs as well. Finally, we show how the proposed parameterization can be enforced efficiently in squared PCs, thus theoretically guaranteeing no loss of expressiveness for certain circuit families (Section 5).

2 Circuits and Squared Circuits

We start by defining circuits in a tensorized formalism (Vergari, Di Mauro, and Esposito 2019; Loconte et al. 2024b).

Definition 1 (Tensorized circuit)**.** A *tensorized circuit* c is a parameterized computational graph encoding a function $c(\mathbf{X})$ and comprising of three kinds of layers: *input*, *product* and *sum*. A layer $\boldsymbol{\ell}$ is a vector-valued function defined over variables $\mathsf{sc}(\boldsymbol{\ell})$, called *scope*, and every non-input layer receives the outputs of other layers as input, denoted as $\mathsf{in}(\boldsymbol{\ell})$. The scope of each non-input layer is the union of the scope of its inputs. The three kinds of layers are defined as follows:

- Each *input layer* $\boldsymbol{\ell}$ has scope $X \in \mathbf{X}$ and computes a collection of K input functions $\{f_i \colon \mathsf{dom}(X) \to \mathbb{C}\}_{i=1}^{K}$, i.e., $\boldsymbol{\ell}$ outputs a K-dimensional vector.
- Each *product layer* $\boldsymbol{\ell}$ computes either an element-wise (or Hadamard) or Kronecker product of its N inputs, i.e., $\odot_{i=1}^{N} \boldsymbol{\ell}_i(\mathsf{sc}(\boldsymbol{\ell}_i))$ or $\otimes_{i=1}^{N} \boldsymbol{\ell}_i(\mathsf{sc}(\boldsymbol{\ell}_i))$, respectively.
- A sum layer $\boldsymbol{\ell}$ with $\boldsymbol{\ell}_1$ as input, is parameterized by $\mathbf{W} \in \mathbb{C}^{K_1 \times K_2}$ and computes the matrix-vector product $\boldsymbol{\ell}(\mathsf{sc}(\boldsymbol{\ell})) = \mathbf{W}\boldsymbol{\ell}_1(\mathsf{sc}(\boldsymbol{\ell}_1))$.

Each non-input layer is a vector-valued function, but each entry it computes is a scalar-valued function computed over certain entries of its input vectors. We denote as $\mathsf{size}(\boldsymbol{\ell})$ the number of scalar inputs to each scalar function encoded in $\boldsymbol{\ell}$. That is, an Hadamard layer consists of K scalar functions each computing the product of N other scalars, thus $\mathsf{size}(\boldsymbol{\ell}) = NK$. A Kronecker layer consists of K^N scalar functions each computing the product of N other scalars, i.e., $\mathsf{size}(\boldsymbol{\ell}) = NK^N$. Finally, a sum layer consists of K_1 scalar functions each receiving input from K_2 other scalars and computing a linear combination, i.e., $\mathsf{size}(\boldsymbol{\ell}) = K_1K_2$. The size of a layer is also its computational complexity.

A tensorized PC is a tensorized circuit c computing non-negative values, i.e., for any $\mathbf{x}$ we have $c(\mathbf{x}) \geq 0$. Thus, a PC c encodes a probability distribution $p(\mathbf{x}) = Z^{-1}c(\mathbf{x})$. A PC c supports tractable marginalization of any variable subset (Choi, Vergari, and Van den Broeck 2020) if (i) the functions encoded by the input layers can be integrated efficiently and (ii) it is *smooth* and *decomposable*.

Definition 2 (Layer-wise smoothness and decomposability (Darwiche and Marquis 2002; Loconte et al. 2024b))**.** A tensorized circuit over variables $\mathbf{X}$ is *smooth* if for every sum layer $\boldsymbol{\ell}$, its inputs depend on all the same variables, i.e., $\forall \boldsymbol{\ell}_i, \boldsymbol{\ell}_j \in \mathsf{in}(\boldsymbol{\ell}) \colon \mathsf{sc}(\boldsymbol{\ell}_i) = \mathsf{sc}(\boldsymbol{\ell}_j)$. It is *decomposable* if for every product layer $\boldsymbol{\ell}$ in it, its inputs depend on disjoint scopes, i.e., $\forall \boldsymbol{\ell}_i, \boldsymbol{\ell}_j \in \mathsf{in}(\boldsymbol{\ell}), i \neq j \colon \mathsf{sc}(\boldsymbol{\ell}_i) \cap \mathsf{sc}(\boldsymbol{\ell}_j) = \varnothing$.

Since sum layers can have only one input in Definition 1, circuits are ensured smooth, but not necessarily decomposable. Popular TNs like MPS (Pérez-García et al. 2007) and tree TNs (Shi, Duan, and Vidal 2006; Cheng et al. 2019) are special cases of smooth and decomposable circuits having a particular tree structure, and use Hadamard and Kronecker layers, respectively (Loconte et al. 2024b). However, Definition 1 allows us to build different factorizations by connecting layers, e.g., mix both Hadamard and Kronecker layers, include different input functions, and share parameters.

Loconte, Mengel, and Vergari (2024) showed one can learn PCs with complex parameters by *squaring* circuits. Given a circuit c that outputs complex numbers, a squared PC c^2 is obtained by multiplying c and its complex conjugate c^*, i.e., $c^2(\mathbf{x}) = |c(\mathbf{x})|^2 = c(\mathbf{x})c^*(\mathbf{x})$. Thus, a squared PC encodes a distribution $p(\mathbf{x}) = Z^{-1}|c(\mathbf{x})|^2$, where $Z = \int_{\mathsf{dom}(\mathbf{X})} |c(\mathbf{x})|^2 \mathrm{d}\mathbf{x}$. Computing Z tractably requires representing c^2 as another decomposable circuit, which can be done if c is *structured-decomposable* (Vergari et al. 2021), i.e., each product layer factorizes its scope to its inputs, and the collection of such factorizations forms a tree (Pipatsrisawat and Darwiche 2008). As detailed in Appendix A, c^2 can be built from c by retaining its structure and by quadratically increasing the size of layers. Thus, computing Z requires time $\mathcal{O}(LS^2)$, where L is the number of layers and S is the maximum layer size in c, i.e., $S = \max_{\boldsymbol{\ell} \in c}\{\mathsf{size}(\boldsymbol{\ell})\}$. In general, marginalizing any variable subset still requires time $\mathcal{O}(LS^2)$ (Loconte et al. 2024a).

Instead, monotonic PCs can be parameterized by (i) using distributions as input functions (e.g., Gaussian), and (ii) normalizing each parameter matrix row, i.e., for each sum layer parameterized by $\mathbf{W} \in \mathbb{R}_{+}^{K_1 \times K_2}$ we have that $\forall i \in [K_1]$ $\sum_{j=1}^{K} w_{ij} = 1$, where we denote $[n] = \{1, \ldots, n\}$. The key advantage of (i,ii) is that that the resulting PC encodes an already-normalized distribution. Moreover, marginalizing any variable subset requires time $\mathcal{O}(LS)$, as they are not squared. Note that (ii) is not restrictive, as it can be enforced efficiently using an algorithm by Peharz et al. (2015).

For squared PCs, *what are the sufficient conditions ensuring they are already normalized?* We present them next, and in Section 4 we show how they lead to an algorithm to compute any marginal that can be much more efficient.

3 Squared Orthonormal Circuits

Our representation of squared PCs is based on the definition of orthonormal circuits we introduce below.

Definition 3 (Orthonormal circuits)**.** A tensorized circuit c over variables $\mathbf{X}$ is said *orthonormal* if (1) each input layer $\boldsymbol{\ell}$ over $X \in \mathbf{X}$ encodes a vector of K orthonormal functions, i.e., $\boldsymbol{\ell}(X) = [f_1(X), \ldots, f_K(X)]^\top$ such that $\forall i, j \in [K]^2 \colon \int_{\mathsf{dom}(X)} f_i(x) f_j(x)^* \mathrm{d}x = \delta_{ij}$, where δ_{ij} denotes the

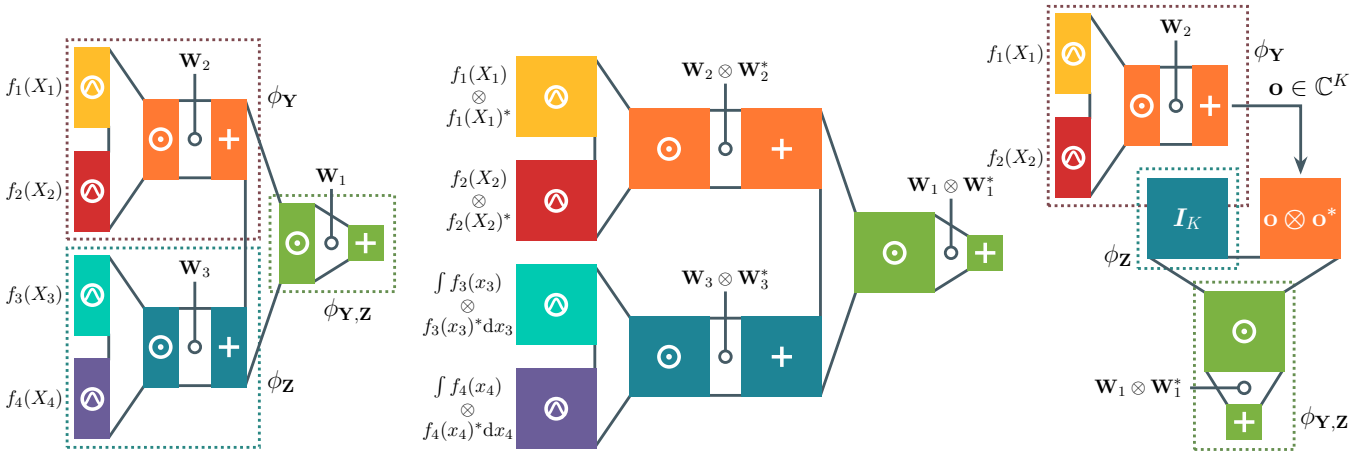

Figure 1: **Squared orthonormal PCs enable a more efficient marginalization algorithm.** The **left** figure shows a tensorized circuit c with a tree structure over $\mathbf{X} = \{X_1, X_2, X_3, X_4\}$ with input , Hadamard and sum layers. We label the input layers with the vector-valued function they encode on a variable X_i. Consider computing the marginal likelihood $p(x_1, x_2) = \int_{\mathsf{dom}(X_3)\times\mathsf{dom}(X_4)} |c(x_1, x_2, x_3, x_4)|^2 \mathrm{d}x_3\mathrm{d}x_4$. We label group of layers depending on $\mathbf{Y} = \{X_1, X_2\}$ (red-ish, $\phi_{\mathbf{Y}}$), $\mathbf{Z} = \{X_3, X_4\}$ (blue-ish, $\phi_{\mathbf{Z}}$), and on both (green, $\phi_{\mathbf{Y},\mathbf{Z}}$). A naive algorithm computing $p(x_1, x_2)$ would (i) square the *whole* tensorized circuit as c^2, where the size of each layer quadratically increases, and (ii) compute the integrals of squared input layers over $\mathbf{Z}$ and (iii) evaluate the rest of the squared layers (**middle**, from left to right). (**right**) Instead, if c is orthonormal, Algorithm B.1 avoids the computation of the integral of the sub-circuit depending on $\mathbf{Z}$ (as it results in the identity matrix $\mathbf{I}_K$, in blue), and requires computing a single Kronecker product (orange) and squaring just the layers in $\phi_{\mathbf{Y},\mathbf{Z}}$ (green).

Kronecker delta; and (2) each sum layer is parameterized by a (semi-)unitary matrix $\mathbf{W} \in \mathbb{C}^{K_1 \times K_2}$, $K_1 \leq K_2$, i.e., $\mathbf{W}\mathbf{W}^\dagger = \mathbf{I}_{K_1}$, or the rows of $\mathbf{W}$ are orthonormal.

If we take a tensorized circuit c that is orthonormal, then the squared PC obtained from c is guaranteed to encode a normalized probability distribution, as formalized below.

Proposition 1. Let c be a structured-decomposable tensorized circuit over variables $\mathbf{X}$. If c is orthonormal, then its squaring encodes a normalized distribution, i.e., $Z = 1$.

Appendix B.1 shows our proof. The idea is that integrating products of input layers encoding orthonormal functions yields identity matrices in c^2. Then, the (semi-)unitarity of parameter matrices in sum layers is used to show the output of each layer in c^2 is (the flattening of) an identity matrix, thus eventually yielding $Z = 1$.

The computation of the partition function Z represents a special case of marginalization, where all variables are marginalized out. In general, computing any marginal probabilities in squared PCs requires worst-case time $\mathcal{O}(LS^2)$ (Loconte et al. 2024a). In the next section, we show how to exploit the properties of orthonormal circuits (Definition 3) to also provide an algorithm that computes any marginal probability with a better complexity.

4 A Tighter Marginalization Complexity

The idea of our algorithm is that, when computing marginal probabilities using c^2, *we do not need to evaluate the layers whose scope depends on only the variables being integrated out.* This is because they would always result in identity matrices, as noticed in our proof for Proposition 1.

In addition, we observe that we do not need to square the *whole* tensorized circuit c, *but only a fraction of the layers depending on* both *the marginalized variables and the ones being kept*. By doing so, a part of the complexity will depend on S rather than S^2. We formalize our result below.

Theorem 1. Let c be a structured-decomposable orthonormal circuit over variables $\mathbf{X}$. Let $\mathbf{Z} \subseteq \mathbf{X}$, $\mathbf{Y} = \mathbf{X} \setminus \mathbf{Z}$. Computing the marginal likelihood $p(\mathbf{y}) = \int_{\mathsf{dom}(\mathbf{Z})} |c(\mathbf{y}, \mathbf{z})|^2 \mathrm{d}\mathbf{z}$ requires time $\mathcal{O}(|\phi_{\mathbf{Y}}|S + |\phi_{\mathbf{Y},\mathbf{Z}}|S^2)$, where $\phi_{\mathbf{Y}}$ (resp. $\phi_{\mathbf{Y},\mathbf{Z}}$) denotes the set of layers in c whose scope depends on only variables in $\mathbf{Y}$ (resp. on variables both in $\mathbf{Y}$ *and* in $\mathbf{Z}$).

We prove it in Appendix B.2 and show our algorithm in Algorithm B.1. Note that the complexity shown in Theorem 1 is independent on the number of layers whose scope depend on $\mathbf{Z}$ only, i.e., $\phi_{\mathbf{Z}}$. Depending on the circuit structure and the variables $\mathbf{Z}$, Algorithm B.1 can be much more efficient than $\mathcal{O}(LS^2)$. For example, the tree structure of a circuit defined over pixel variables can be built by recursively splitting an image into patches with horizontal and vertical cuts (Loconte et al. 2024b). If $\mathbf{Z}$ consists of only the pixel variables of the left-hand side of an image (i.e., we are computing the marginal of the right-hand side $\mathbf{Y}$), then $|\phi_{\mathbf{Y},\mathbf{Z}}| \ll L$ since only a few layers near the root will depend on both $\mathbf{Y}$ and $\mathbf{Z}$. We illustrate an example in Fig. 1.

5 Are Orthonormal Circuits less Expressive?

Orthonormal tensorized circuits restrict their input layers to encode orthonormal functions, and require parameter matrices to be (semi-)unitary (Definition 3), thus arising the question whether these conditions make them less expres-

sive when compared to non-orthonormal ones. Below, we start by analyzing which input layer functions we can encode in terms of orthonormal functions.

Are orthonormal functions restrictive? Depending on whether a variable is discrete or continuous, we have different ways to model it with orthonormal functions. For a discrete variable X with finite domain $\mathsf{dom}(X) = [v]$, any function $f(X)$ can be expressed as $f(x) = \sum_{k=1}^{v} f(k)\delta_{xk}$, i.e., f can be written in terms of v basis functions $\{\delta_{xk}\}_{k=1}^{v}$ that are orthonormal. That is, we have that $\sum_{x\in\mathsf{dom}(X)} \delta_{xk}\delta_{xk'} = \delta_{kk'}$ for $k, k' \in [v]$. Therefore, any input layer over a finitely-discrete variable X can be exactly encoded with a sum layer having a layer encoding the orthonormal basis $\{\delta_{xk}\}_{k=1}^{v}$ as input.

In the case of a continuous variable X, many classes of functions can be expressed in terms of orthonormal basis functions. For instance, periodic functions can be represented by Fourier series of sines and cosines basis that form an orthonormal set of functions (Jackson 1941). Under certain continuity conditions, functions can be approximated arbitrarily well by finite Fourier partial sums (Jackson 1982). Moreover, depending on the support of X, many classes of functions can be described in terms of families of orthonormal polynomials, such as Legendre, Laguerre and Hermite polynomials (Abramowitz, Stegun, and Miller 1965). Notably, Hermite functions generalize Gaussians and are a basis of square-integrable functions over all $\mathbb{R}$ (Roman 1984).

Are (semi-)unitary matrices restrictive? Next, we investigate whether the requirement of sum layer parameter matrices to be (semi-)unitary may reduce the expressiveness of squared PCs. In the following, we answer to this question negatively, as we can enforce this condition in polynomial time w.r.t the number of layers and the layer size.

Theorem 2. Let c be a tensorized circuit over variables $\mathbf{X}$. Assume that each input layer in c encodes a set of orthonormal functions. Then, there exists an algorithm returning an orthonormal circuit c' in polynomial time such that c' is equivalent to c up to a multiplicative constant, i.e., $c'(\mathbf{X}) = Z^{-\frac{1}{2}} c(\mathbf{X})$ where $Z = \int_{\mathsf{dom}(\mathbf{X})} |c(\mathbf{x})|^2 \mathrm{d}\mathbf{x}$.

Appendix B.3 presents our proof, and Algorithm 1 shows our algorithm to "*orthonormalize*" a tensorized circuit. The idea of Algorithm 1 is that we can recursively make subcircuits orthonormal by (i) factorizing the sum layer parameter matrices via QR decompositions, and (ii) by "*pushing*" the non-unitary part of such a decomposition towards the output, until the reciprocal square root of the partition function of c^2 is retrieved at the top level of the recursion. Fig. B.1 illustrates the algorithm. Therefore, Theorem 2 guarantees that squared orthonormal PCs are as expressive as general squared PCs with orthonormal input functions.

Finally, we note that Algorithm 1 is the dual of a result about monotonic PCs shown by Peharz et al. (2015): they show an algorithm that updates the positive weights of a smooth and decomposable PC such that the distribution it encodes is already normalized. Here, we show an algorithm that updates the (possibly) complex weights of a structured-decomposable circuit such that its squaring encodes an already-normalized probability distribution.

Algorithm 1: ORTHONORMALIZE($\boldsymbol{\ell}$)

Input: The output layer $\boldsymbol{\ell}$ of a structured-decomposable circuit, whose input layers encode a set of orthonormal functions.
Output: The output layer of a structured-decomposable circuit c' that is orthonormal, and a matrix $\mathbf{R} \in \mathbb{C}^{K_1 \times K_2}$, $K_1 \leq K_2$.

if $\boldsymbol{\ell}$ is an input layer **then**
  Assume $\boldsymbol{\ell}$ outputs vectors in $\mathbb{C}^K$
  return $(\boldsymbol{\ell}, \mathbf{I}_K)$
if $\boldsymbol{\ell}$ is a sum layer with input $\boldsymbol{\ell}_1$ and parameterized by $\mathbf{W}$ **then**
  $(\boldsymbol{\ell}'_1, \mathbf{R}_1) \leftarrow$ ORTHONORMALIZE$(\boldsymbol{\ell}_1)$, $\mathbf{R}_1 \in \mathbb{C}^{K_2 \times K_3}$
  Assume $\mathbf{W} \in \mathbb{C}^{K_1 \times K_2}$, $K_1 \leq K_2$
  Let $\mathbf{V} = \mathbf{W}\mathbf{R}_1 \in \mathbb{C}^{K_1 \times K_3}$, $K_1 \leq K_3$
  Factorize $\mathbf{V}^\dagger = \mathbf{Q}\mathbf{R}$ where $\mathbf{Q} \in \mathbb{C}^{K_3 \times K_1}$, $\mathbf{R} \in \mathbb{C}^{K_1 \times K_1}$
    such that $\mathbf{Q}^\dagger\mathbf{Q} = \mathbf{I}_{K_1}$ and $\mathbf{R}$ is upper triangular
  Let $\boldsymbol{\ell}'$ be a sum layer computing $\mathbf{Q}^\dagger \boldsymbol{\ell}'_1(\mathsf{sc}(\boldsymbol{\ell}))$
  return $(\boldsymbol{\ell}', \mathbf{R}^\dagger)$
if $\boldsymbol{\ell}$ is a Kronecker product layer with inputs $\boldsymbol{\ell}_1, \boldsymbol{\ell}_2$ **then**
  $(\boldsymbol{\ell}'_1, \mathbf{R}_1) \leftarrow$ ORTHONORMALIZE$(\boldsymbol{\ell}_1)$, $\mathbf{R}_1 \in \mathbb{C}^{K_1 \times K_2}$
  $(\boldsymbol{\ell}'_2, \mathbf{R}_2) \leftarrow$ ORTHONORMALIZE$(\boldsymbol{\ell}_2)$, $\mathbf{R}_2 \in \mathbb{C}^{K_3 \times K_4}$
  Let $\boldsymbol{\ell}'$ be a layer computing $\boldsymbol{\ell}'_1(\mathsf{sc}(\boldsymbol{\ell}_1)) \otimes \boldsymbol{\ell}'_2(\mathsf{sc}(\boldsymbol{\ell}_2))$
  return $(\boldsymbol{\ell}', \mathbf{R}_1 \otimes \mathbf{R}_2)$ ▷ $\otimes$: Kronecker matrix product
if $\boldsymbol{\ell}$ is an Hadamard product layer with inputs $\boldsymbol{\ell}_1, \boldsymbol{\ell}_2$ **then**
  $(\boldsymbol{\ell}'_1, \mathbf{R}_1) \leftarrow$ ORTHONORMALIZE$(\boldsymbol{\ell}_1)$, $\mathbf{R}_1 \in \mathbb{C}^{K_1 \times K_2}$
  $(\boldsymbol{\ell}'_2, \mathbf{R}_2) \leftarrow$ ORTHONORMALIZE$(\boldsymbol{\ell}_2)$, $\mathbf{R}_2 \in \mathbb{C}^{K_1 \times K_3}$
  Let $\boldsymbol{\ell}'$ be a layer computing $\boldsymbol{\ell}'_1(\mathsf{sc}(\boldsymbol{\ell}_1)) \otimes \boldsymbol{\ell}'_2(\mathsf{sc}(\boldsymbol{\ell}_2))$
  return $(\boldsymbol{\ell}', \mathbf{R}_1 \bullet \mathbf{R}_2)$ ▷ $\bullet$: Face-splitting matrix product

6 Related Work and Conclusion

In this paper, we presented a parameterization of squared PCs inspired by canonical forms in TNs, based on orthonormal functions and (semi-)unitary matrices, as to speed-up the computation of marginal probabilities. As squared orthonormal PCs support faster marginalization, they are amenable for future works on applications where computing marginals is key, e.g., lossless compression (Yang, Mandt, and Theis 2022; Liu, Mandt, and Van den Broeck 2022).

Orthonormal basis functions have been used to parameterize *shallow* squared mixtures encoding already-normalized distributions, both in signal processing (Pinheiro and Vidakovic 1997) and in score-based variational inference (Cai et al. 2024). Our squared orthonormal PCs generalize them, as they represent *deeper* squared mixtures.

We plan to investigate different methods as to learn squared orthonormal PCs from data for distribution estimation, and compare how do they perform w.r.t. squared PCs with unconstrained parameters. For instance, there are many ways of parameterizing unitary matrices, with different advantages regarding efficiency, numerical stability, and optimization (Arjovsky, Shah, and Bengio 2015; Huang et al. 2017; Bansal, Chen, and Wang 2018; Casado and Martínez-Rubio 2019). Moreover, Hauru, Damme, and Haegeman (2020); Luchnikov et al. (2021) proposed optimizing the parameters of MPS TNs and quantum gates by performing gradient descent over the Stiefel manifold (Absil, Mahony, and Sepulchre 2007). Furthermore, recent works parameterize more expressive monotonic PCs using neural networks (Shao et al. 2022; Gala et al. 2024a,b), thus motivating parameterizing squared orthonormal PCs similarly.

Acknowledgments

We acknowledge Raul Garcia-Patron Sanchez for meaningful discussions about tensor networks and quantum circuits. AV was supported by the "UNREAL: Unified Reasoning Layer for Trustworthy ML" project (EP/Y023838/1) selected by the ERC and funded by UKRI EPSRC.

Contributions

LL and AV conceived the initial idea of the paper. LL is responsible for all theoretical contributions, pictures, algorithms and writing. AV supervised all the phases of the project and provided feedback.

References

Abramowitz, M.; Stegun, I. A.; and Miller, D. 1965. Handbook of Mathematical Functions With Formulas, Graphs and Mathematical Tables (National Bureau of Standards Applied Mathematics Series No. 55). *Journal of Applied Mechanics*, 32: 239–239.

Absil, P.-A.; Mahony, R. E.; and Sepulchre, R. 2007. *Optimization Algorithms on Matrix Manifolds*. Princeton University Press.

Ahmed, K.; Teso, S.; Chang, K.-W.; Van den Broeck, G.; and Vergari, A. 2022. Semantic probabilistic layers for neuro-symbolic learning. In *Advances in Neural Information Processing Systems 35 (NeurIPS)*, volume 35, 29944–29959. Curran Associates, Inc.

Arjovsky, M.; Shah, A.; and Bengio, Y. 2015. Unitary Evolution Recurrent Neural Networks. In *International Conference on Machine Learning*.

Bansal, N.; Chen, X.; and Wang, Z. 2018. Can We Gain More from Orthogonality Regularizations in Training Deep Networks? In *Advances in Neural Information Processing Systems*, volume 31.

Biamonte, J. D.; and Bergholm, V. 2017. Tensor Networks in a Nutshell. *arXiv: Quantum Physics*.

Cai, D.; Modi, C.; Margossian, C.; Gower, R. M.; Blei, D.; and Saul, L. K. 2024. EigenVI: score-based variational inference with orthogonal function expansions. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems (NeurIPS)*.

Casado, M. L.; and Martínez-Rubio, D. 2019. Cheap Orthogonal Constraints in Neural Networks: A Simple Parametrization of the Orthogonal and Unitary Group. *ArXiv*, abs/1901.08428.

Cheng, S.; Wang, L.; Xiang, T.; and Zhang, P. 2019. Tree tensor networks for generative modeling. *Physical Review B*, 99(15): 155131.

Choi, Y.; Vergari, A.; and Van den Broeck, G. 2020. Probabilistic Circuits: A Unifying Framework for Tractable Probabilistic Modeling. Technical report, University of California, Los Angeles (UCLA).

Darwiche, A.; and Marquis, P. 2002. A knowledge compilation map. *Journal of Artificial Intelligence Research (JAIR)*, 17: 229–264.

Gala, G.; de Campos, C.; Peharz, R.; Vergari, A.; and Quaeghebeur, E. 2024a. Probabilistic Integral Circuits. In *AISTATS 2024*.

Gala, G.; de Campos, C.; Vergari, A.; and Quaeghebeur, E. 2024b. Scaling Continuous Latent Variable Models as Probabilistic Integral Circuits. *arXiv preprint arXiv:2406.06494*.

Glasser, I.; Pancotti, N.; and Cirac, J. I. 2018. From Probabilistic Graphical Models to Generalized Tensor Networks for Supervised Learning. *IEEE Access*, 8: 68169–68182.

Glasser, I.; Sweke, R.; Pancotti, N.; Eisert, J.; and Cirac, I. 2019. Expressive power of tensor-network factorizations for probabilistic modeling. In *Advances in Neural Information Processing Systems 32 (NeurIPS)*, 1498–1510. Curran Associates, Inc.

Hauru, M.; Damme, M. V.; and Haegeman, J. 2020. Riemannian optimization of isometric tensor networks. *SciPost Physics*.

Huang, L.; Liu, X.; Lang, B.; Yu, A. W.; and Li, B. 2017. Orthogonal Weight Normalization: Solution to Optimization over Multiple Dependent Stiefel Manifolds in Deep Neural Networks. In *AAAI Conference on Artificial Intelligence*.

Jackson, D. 1941. Fourier series and orthogonal polynomials.

Jackson, D. 1982. The theory of approximation.

Liu, A.; Mandt, S.; and Van den Broeck, G. 2022. Lossless Compression with Probabilistic Circuits. In *International Conference on Learning Representations*.

Loconte, L.; Aleksanteri, M. S.; Mengel, S.; Trapp, M.; Solin, A.; Gillis, N.; and Vergari, A. 2024a. Subtractive Mixture Models via Squaring: Representation and Learning. In *The Twelfth International Conference on Learning Representations (ICLR)*.

Loconte, L.; Di Mauro, N.; Peharz, R.; and Vergari, A. 2023. How to Turn Your Knowledge Graph Embeddings into Generative Models via Probabilistic Circuits. In *Advances in Neural Information Processing Systems 37 (NeurIPS)*. Curran Associates, Inc.

Loconte, L.; Mari, A.; Gala, G.; Peharz, R.; de Campos, C.; Quaeghebeur, E.; Vessio, G.; and Vergari, A. 2024b. What is the Relationship between Tensor Factorizations and Circuits (and How Can We Exploit it)? arXiv:2409.07953.

Loconte, L.; Mengel, S.; and Vergari, A. 2024. Sum of Squares Circuits. arXiv:2408.11778.

Luchnikov, I. A.; Ryzhov, A.; Filippov, S. N.; and Ouerdane, H. 2021. QGOpt: Riemannian optimization for quantum technologies. *SciPost Physics*.

Novikov, G. S.; Panov, M. E.; and Oseledets, I. V. 2021. Tensor-train density estimation. In *37th Conference on Uncertainty in Artificial Intelligence (UAI)*, volume 161 of *Proceedings of Machine Learning Research*, 1321–1331. PMLR.

Orús, R. 2013. A Practical Introduction to Tensor Networks: Matrix Product States and Projected Entangled Pair States. *Annals of Physics*, 349: 117–158.

Peharz, R.; Tschiatschek, S.; Pernkopf, F.; and Domingos, P. M. 2015. On Theoretical Properties of Sum-Product Networks. In *International Conference on Artificial Intelligence and Statistics*.

Pérez-García, D.; Verstraete, F.; Wolf, M. M.; and Cirac, J. I. 2007. Matrix Product State Representations. *Quantum Information and Computing*, 7(5): 401–430.

Pinheiro, A.; and Vidakovic, B. 1997. Estimating the square root of a density via compactly supported wavelets. *Computational Statistics and Data Analysis*, 25(4): 399–415.

Pipatsrisawat, K.; and Darwiche, A. 2008. New Compilation Languages Based on Structured Decomposability. In *23rd Conference on Artificial Intelligence (AAAI)*, volume 8, 517–522.

Roman, S. 1984. The Umbral Calculus.

Schollwoeck, U. 2010. The density-matrix renormalization group in the age of matrix product states. *Annals of Physics*, 326: 96–192.

Shao, X.; Molina, A.; Vergari, A.; Stelzner, K.; Peharz, R.; Liebig, T.; and Kersting, K. 2022. Conditional sum-product networks: Modular probabilistic circuits via gate functions. *International Journal of Approximate Reasoning*, 140: 298–313.

Shi, Y.-Y.; Duan, L.-M.; and Vidal, G. 2006. Classical simulation of quantum many-body systems with a tree tensor network. *Physical Review A*, 74: 22320.

Shpilka, A.; and Yehudayoff, A. 2010. Arithmetic Circuits: A survey of recent results and open questions. *Founddations and Trends in Theoretical Computer Science*, 5: 207–388.

Stoudenmire, E.; and Schwab, D. J. 2016. Supervised Learning with Tensor Networks. In *Advances in Neural Information Processing Systems 29 (NeurIPS)*, 4799–4807. Curran Associates, Inc.

Vergari, A.; Choi, Y.; Liu, A.; Teso, S.; and Van den Broeck, G. 2021. A Compositional Atlas of Tractable Circuit Operations for Probabilistic Inference. In *Advances in Neural Information Processing Systems 34 (NeurIPS)*, 13189–13201. Curran Associates, Inc.

Vergari, A.; Di Mauro, N.; and Esposito, F. 2019. Visualizing and understanding sum-product networks. *Machine Learning*, 108(4): 551–573.

Yang, Y.; Mandt, S.; and Theis, L. 2022. An Introduction to Neural Data Compression. *Foundations and Trends in Computer Graphics and Vision*, 15: 113–200.

Zhang, H.; Dang, M.; Peng, N.; and Van den Broeck, G. 2023. Tractable Control for Autoregressive Language Generation. In *40th International Conference on Machine Learning (ICML)*, volume 202 of *Proceedings of Machine Learning Research*, 40932–40945. PMLR.

Algorithm A.1: SQUARETENSORIZEDCIRCUIT($\boldsymbol{\ell}$)

Input: A tensorized circuit with output layer $\boldsymbol{\ell}$ that is structured-decomposable.

Output: The tensorized squared circuit having $\boldsymbol{\ell}^2$ as output layer computing $\boldsymbol{\ell} \otimes \boldsymbol{\ell}^*$.

1: **if** $\boldsymbol{\ell}$ is an input layer **then**
2: $\quad$ $\boldsymbol{\ell}$ computes K functions $\{f_i\}_{i=1}^{K}$ over X
3: $\quad$ **return** An input layer $\boldsymbol{\ell}^2$ computing all K^2
4: $\qquad$ product combinations $f_i(X)f_j^*(X)$
5: **if** $\boldsymbol{\ell}$ is a product layer **then**
6: $\quad$ $(\boldsymbol{\ell}_1, \boldsymbol{\ell}_2) \leftarrow$ GETINPUTS($\boldsymbol{\ell}$)
7: $\quad$ $\boldsymbol{\ell}_1^2 \leftarrow$ SQUARETENSORIZEDCIRCUIT($\boldsymbol{\ell}_1$)
8: $\quad$ $\boldsymbol{\ell}_2^2 \leftarrow$ SQUARETENSORIZEDCIRCUIT($\boldsymbol{\ell}_2$)
9: $\quad$ **if** $\boldsymbol{\ell}$ is an Hadamard product layer **then**
10: $\qquad$ **return** $\boldsymbol{\ell}_1^2(\mathsf{sc}(\boldsymbol{\ell}_1)) \odot \boldsymbol{\ell}_2^2(\mathsf{sc}(\boldsymbol{\ell}_2))$
11: $\quad$ **else** $\qquad$ ▷ $\boldsymbol{\ell}$ is Kronecker product layer
12: $\qquad$ **return** $\mathbf{P}\left(\boldsymbol{\ell}_1^2(\mathsf{sc}(\boldsymbol{\ell}_1)) \otimes \boldsymbol{\ell}_2^2(\mathsf{sc}(\boldsymbol{\ell}_2))\right)$,
13: $\qquad\quad$ where $\mathbf{P}$ is a permutation matrix
$\qquad\qquad$ ▷ $\boldsymbol{\ell}$ is a sum layer
14: $(\boldsymbol{\ell}_1,) \leftarrow$ GETINPUTS($\boldsymbol{\ell}$)
15: $\boldsymbol{\ell}_1^2 \leftarrow$ SQUARETENSORIZEDCIRCUIT($\boldsymbol{\ell}_1$)
16: $\mathbf{W} \in \mathbb{R}^{S \times K} \leftarrow$ GETPARAMETERS($\boldsymbol{\ell}$)
17: $\mathbf{W}' \in \mathbb{R}^{S^2 \times K^2} \leftarrow \mathbf{W} \otimes \mathbf{W}^*$
18: **return** $\mathbf{W}'\boldsymbol{\ell}_1^2(\mathsf{sc}(\boldsymbol{\ell}_1))$

A Squaring Tensorized Circuits

Given a tensorized circuit c, a squared PC models $p(\mathbf{X}) = Z^{-1}|c(\mathbf{X})|^2 = Z^{-1}c(\mathbf{X})c(\mathbf{X})^*$. To compute Z efficiently, one has to represent $|c(\mathbf{X})|^2$ as a decomposable circuit (Definition 2), since it would allow tractable variable marginalization (Choi, Vergari, and Van den Broeck 2020).

Algorithm A.1 recursively constructs the circuit c^2 computing $|c(\mathbf{X})|^2$ as yet another decomposable tensorized circuit. Algorithm A.1 is taken from Loconte et al. (2024a), but here we trivially generalize it as to allow complex weight parameter matrices. In this algorithm, each layer $\boldsymbol{\ell}$ in c is recursively squared into a layer $\boldsymbol{\ell}^2$ in c^2 as to compute the Kronecker product between the output of $\boldsymbol{\ell}$ and its conjugate. As a consequence, the size of each layer in c is quadratically increased in c^2. For instance, each input layer $\boldsymbol{\ell}$ in c over a variable X and computing $\boldsymbol{\ell}(X) \in \mathbb{C}^K$ is squared as to recover an input layer $\boldsymbol{\ell}^2$ in c^2 such that $\boldsymbol{\ell}^2(X) = \boldsymbol{\ell}(X) \otimes \boldsymbol{\ell}(X)^* \in \mathbb{C}^{K^2}$ (L1-4). An Hadamard (resp. Kronecker) product layer in c is squared as another Hadamard layer (resp. a composition of sum and Kronecker layers) in c^2 (L5-13). Finally, a sum layer $\boldsymbol{\ell}$ in c computing $\mathbf{W}\boldsymbol{\ell}_1(\mathsf{sc}(\boldsymbol{\ell}_1))$ is squared as to recover a sum layer $\boldsymbol{\ell}^2$ in c^2 that is parameterized by $\mathbf{W} \otimes \mathbf{W}^*$ instead (L14-18). Fig. 1 (left) shows an example of tensorized circuit and Fig. 1 (middle) shows its squaring obtained with Algorithm A.1.

B Proofs

We start with some notation.

Notation. In Proposition 1 and Theorem 1 we require integrating layers $\boldsymbol{\ell}$ that output vectors, e.g., in $\mathbb{C}^K$. That is, given a layer $\boldsymbol{\ell}$ having scope $\mathsf{sc}(\boldsymbol{\ell}) = \mathbf{Y} \cup \mathbf{Z}$ and encoding a function $\boldsymbol{\ell}\colon \mathsf{dom}(\mathbf{Y} \cup \mathbf{Z}) \to \mathbb{C}^K$, we write $\int_{\mathsf{dom}(\mathbf{Z})} \boldsymbol{\ell}(\mathbf{y}, \mathbf{z})\mathrm{d}\mathbf{z}$ to refer to the K-dimensional vector obtained by integrating the K function components encoded by $\boldsymbol{\ell}$, i.e., $\int_{\mathsf{dom}(\mathbf{Z})} \boldsymbol{\ell}(\mathbf{y}, \mathbf{z})\mathrm{d}\mathbf{z} =$

$$= \left[\int_{\mathsf{dom}(\mathbf{Z})} \boldsymbol{\ell}(\mathbf{y}, \mathbf{z})_1\mathrm{d}\mathbf{z} \quad \cdots \quad \int_{\mathsf{dom}(\mathbf{Z})} \boldsymbol{\ell}(\mathbf{y}, \mathbf{z})_K\mathrm{d}\mathbf{z}\right]^\top \in \mathbb{C}^K.$$

Therefore, due to the linearity of the function computed by sum layers, we write $\int_{\mathsf{dom}(\mathbf{Z})} \mathbf{W}\boldsymbol{\ell}(\mathbf{y}, \mathbf{z})\mathrm{d}\mathbf{z} =$

$$\begin{aligned} &= \mathbf{W}\left[\int_{\mathsf{dom}(\mathbf{Z})} \boldsymbol{\ell}(\mathbf{y}, \mathbf{z})_1\mathrm{d}\mathbf{z} \quad \cdots \quad \int_{\mathsf{dom}(\mathbf{Z})} \boldsymbol{\ell}(\mathbf{y}, \mathbf{z})_K\mathrm{d}\mathbf{z}\right]^\top \\ &= \mathbf{W}\int_{\mathsf{dom}(\mathbf{Z})} \boldsymbol{\ell}(\mathbf{y}, \mathbf{z})\mathrm{d}\mathbf{z}. \end{aligned}$$

For Hadamard product layers in decomposable circuits (Definition 2), we generally write $\int_{\mathsf{dom}(\mathbf{Z}_1)\times\mathsf{dom}(\mathbf{Z}_2)} \boldsymbol{\ell}_1(\mathbf{y}_1, \mathbf{z}_1) \odot \boldsymbol{\ell}_2(\mathbf{y}_2, \mathbf{z}_2)\mathrm{d}\mathbf{z}_1\mathrm{d}\mathbf{z}_2 =$

$$\int_{\mathsf{dom}(\mathbf{Z}_1)} \boldsymbol{\ell}_1(\mathbf{y}_1, \mathbf{z}_1)\mathrm{d}\mathbf{z}_1 \odot \int_{\mathsf{dom}(\mathbf{Z}_2)} \boldsymbol{\ell}_2(\mathbf{y}_1, \mathbf{z}_2)\mathrm{d}\mathbf{z}_2,$$

where $(\mathbf{Y}_1, \mathbf{Y}_2)$ is a partitioning of $\mathbf{Y}$ and $(\mathbf{Z}_1, \mathbf{Z}_2)$ is a partitioning of $\mathbf{Z}$. Similarly, we will write $\int_{\mathsf{dom}(\mathbf{Z}_1)\times\mathsf{dom}(\mathbf{Z}_2)} \boldsymbol{\ell}_1(\mathbf{y}_1, \mathbf{z}_1) \otimes \boldsymbol{\ell}_2(\mathbf{y}_2, \mathbf{z}_2)\mathrm{d}\mathbf{z}_1\mathrm{d}\mathbf{z}_2$ as above by replacing $\odot$ with $\otimes$ instead.

B.1 Normalized Squared Circuits

Proposition 1. Let c be a structured-decomposable tensorized circuit over variables $\mathbf{X}$. If c is orthonormal, then its squaring encodes a normalized distribution, i.e., $Z = 1$.

Proof. We prove it by showing how the orthonormal property satisfied by c (Definition 3) yields $Z = 1$. In particular, we do so by following Algorithm A.1 to compute the modulus square of a tensorized circuit c that is structured-decomposable as yet another smooth and decomposable circuit c^2. The idea is to recursively show that the output of each layer in c^2 must output the flattening (or vectorization) of an identity matrix when computing Z, thus yielding $Z = 1$ as output in the last step of the recursion.

Case (i): input layer. Given an input layer $\boldsymbol{\ell}$ computing a vector of K orthonormal functions $\boldsymbol{\ell}(X) = [f_1(X), \ldots, f_K(X)]^\top$, Algorithm A.1 materializes another input layer $\boldsymbol{\ell}^2$ such that $\boldsymbol{\ell}^2(X) = \boldsymbol{\ell}(X) \otimes \boldsymbol{\ell}(X)^*$ (L3-4). Thus, $\boldsymbol{\ell}^2$ computes a vector of K^2 functions $\{f_i(X)f_j^*(X) \mid i, j \in [K]\}$, and it is an input layer of the squared PC c^2. Since $\boldsymbol{\ell}$ encodes orthonormal functions, we observe that integrating $\boldsymbol{\ell}^2$ over the whole domain of X yields $\int_{\mathsf{dom}(\mathbf{Z})} \boldsymbol{\ell}^2(\mathbf{z})\mathrm{d}\mathbf{z} = \mathsf{vec}(\mathbf{I}_K)$, where $\mathsf{vec}(\,\cdot\,)$ denotes the flattening of a matrix into a vector.

Case (ii): Hadamard product layer. Given a Hadamard product layer $\boldsymbol{\ell}$ with scope $\mathbf{Z}$ and computing $\boldsymbol{\ell}_1(\mathbf{Z}_1) \odot \boldsymbol{\ell}_2(\mathbf{Z}_2)$ with $\mathbf{Z}_1 \cap \mathbf{Z}_2 = \varnothing$, $\mathbf{Z}_1 \cup \mathbf{Z}_2 = \mathbf{Z}$, Algorithm A.1 constructs another Hadamard product layer $\boldsymbol{\ell}^2$

computing $\ell_1^2(\mathbf{Z}_1) \odot \ell_2^2(\mathbf{Z}_2)$, where ℓ_1^2 (resp. ℓ_2^2) is the squaring of the layer ℓ_1 (resp. ℓ_2) obtained recursively in Algorithm A.1 (L8-9). Now, if $\int_{\mathsf{dom}(\mathbf{Z}_1)} \ell_1^2(\mathbf{z}_1)\mathrm{d}\mathbf{z}_1 = \mathsf{vec}(\mathbf{I}_{K_\ell})$ and $\int_{\mathsf{dom}(\mathbf{Z}_2)} \ell_2^2(\mathbf{z}_2)\mathrm{d}\mathbf{z}_2 = \mathsf{vec}(\mathbf{I}_{K_\ell})$, then we have that

$$\int_{\mathsf{dom}(\mathbf{Z})} \ell^2(\mathbf{z})\mathrm{d}\mathbf{z} = \mathsf{vec}(\mathbf{I}_{K_\ell}) \odot \mathsf{vec}(\mathbf{I}_{K_\ell}) = \mathsf{vec}(\mathbf{I}_{K_\ell}),$$

by exploiting the decomposability of c (thus also c^2 (Vergari et al. 2021)) (Definition 2).

Case (iii): Kronecker product layer. Given a Kronecker product layer ℓ with scope $\mathbf{Z}$ and computing $\ell_1(\mathbf{Z}_1) \otimes \ell_2(\mathbf{Z}_2)$ with $\mathbf{Z}_1 \cap \mathbf{Z}_2 = \varnothing$, $\mathbf{Z}_1 \cup \mathbf{Z}_2 = \mathbf{Z}$, Algorithm A.1 constructs a composition of a sum and a Kronecker product layer $\ell^2(\mathbf{Z}) = \mathbf{P}(\ell_1^2(\mathbf{Z}_1) \otimes \ell_2^2(\mathbf{Z}_2))$, where $\mathbf{P}$ is a permutation matrix ensuring ℓ^2 outputs $\ell(\mathbf{Z}) \otimes \ell(\mathbf{Z})$ (as the Kronecker product is not commutative). Similarly to the Hadamard product layer above, if we assume by inductive hypothesis that $\int_{\mathsf{dom}(\mathbf{Z}_1)} \ell_1^2(\mathbf{z}_1)\mathrm{d}\mathbf{z}_1 = \mathsf{vec}(\mathbf{I}_{K_{\ell_1}})$ and $\int_{\mathsf{dom}(\mathbf{Z}_2)} \ell_2^2(\mathbf{z}_2)\mathrm{d}\mathbf{z}_2 = \mathsf{vec}(\mathbf{I}_{K_{\ell_2}})$, then we recover that

$$\begin{aligned}\int_{\mathsf{dom}(\mathbf{Z})} \ell^2(\mathbf{z})\mathrm{d}\mathbf{z} &= \mathbf{P}(\mathsf{vec}(\mathbf{I}_{K_{\ell_1}}) \otimes \mathsf{vec}(\mathbf{I}_{K_{\ell_2}})) \\ &= \mathsf{vec}(\mathbf{I}_{K_\ell}),\end{aligned}$$

where $K_\ell = K_{\ell_1} \cdot K_{\ell_2}$, by again exploiting the decomposability of c.

Case (iv): sum layer. Finally, let ℓ be a sum layer over $\mathbf{Z}$ and computing the matrix-vector product $\mathbf{W}\ell_1(\mathbf{Z})$, with $\mathbf{W} \in \mathbb{C}^{K_1 \times K_2}$, $K_1 \leq K_2$ and $\mathbf{W}\mathbf{W}^\dagger = \mathbf{I}_{K_1}$ by hypothesis. Algorithm A.1 materializes a sum layer ℓ^2 computing $\ell^2(\mathbf{Z}) = (\mathbf{W} \otimes \mathbf{W}^*)\ell_1^2(\mathbf{Z})$, where ℓ_1^2 is the squared layer obtained from ℓ_1 by recursion (L15). Now, if we assume that $\int_{\mathsf{dom}(\mathbf{Z})} \ell_1^2(\mathbf{z})\mathrm{d}\mathbf{z} = \mathsf{vec}(\mathbf{I}_{K_2})$, then we have that

$$\begin{aligned}\int_{\mathsf{dom}(\mathbf{Z})} \ell^2(\mathbf{z})\mathrm{d}\mathbf{z} &= (\mathbf{W} \otimes \mathbf{W}^*)\mathsf{vec}(\mathbf{I}_{K_2}) \\ &= \mathsf{vec}(\mathbf{W}\mathbf{I}_{K_2}\mathbf{W}^\dagger) = \mathsf{vec}(\mathbf{I}_{K_1}).\end{aligned}$$

Therefore, if ℓ (resp. ℓ^2) is the output layer of the tensorized circuit c (resp. c^2), then $\mathbf{Z} = \mathbf{X}$, $K_1 = 1$ as ℓ must output a scalar, and therefore $Z = \int_{\mathsf{dom}(\mathbf{X})} |c(\mathbf{x})|^2\mathrm{d}\mathbf{x} = 1$. □

B.2 A Tighter Marginalization Complexity

Theorem 1. Let c be a structured-decomposable orthonormal circuit over variables $\mathbf{X}$. Let $\mathbf{Z} \subseteq \mathbf{X}$, $\mathbf{Y} = \mathbf{X} \setminus \mathbf{Z}$. Computing the marginal likelihood $p(\mathbf{y}) = \int_{\mathsf{dom}(\mathbf{Z})} |c(\mathbf{y}, \mathbf{z})|^2\mathrm{d}\mathbf{z}$ requires time $\mathcal{O}(|\phi_{\mathbf{Y}}|S + |\phi_{\mathbf{Y},\mathbf{Z}}|S^2)$, where $\phi_{\mathbf{Y}}$ (resp. $\phi_{\mathbf{Y},\mathbf{Z}}$) denotes the set of layers in c whose scope depends on only variables in $\mathbf{Y}$ (resp. on variables both in $\mathbf{Y}$ *and* in $\mathbf{Z}$).

Proof. We prove it by constructing Algorithm B.1, i.e., the algorithm computing the marginal likelihood given by hypothesis. Algorithm B.1 is based on two ideas. First, integrating sub-circuits whose layer depend only on the variables being integrated over (i.e., $\mathbf{Z}$) will yield identity matrices, so there is no need to evaluate them. Second, the sub-circuits whose layers depend on the variables that are *not* integrated over (i.e., $\mathbf{Y}$) do not need to be squared and can be evaluated without squaring their size (see Appendix A).

Below, we consider different cases for each layer and based on the variables they depend on.

Case (i) layers depending on variables Z only. Consider a layer ℓ in c such that $\mathsf{sc}(\ell) \subseteq \mathbf{Z}$, i.e., $\ell \in \phi_{\mathbf{Z}}$ by hypothesis. Moreover, let $\mathbf{Z}' = \mathbf{Z} \cap \mathsf{sc}(\ell)$. Since the sub-circuit rooted in ℓ is orthonormal by hypothesis, we have that integrating such a sub-circuit yields the flattening of an identity matrix, i.e., $\int_{\mathsf{dom}(\mathbf{Z}')} \ell^2(\mathbf{z}')\mathrm{d}\mathbf{z}' = \mathsf{vec}(\mathbf{I}_K)$, where ℓ^2 is the squared layer constructed by Algorithm A.1, and K is the size of the outputs of ℓ. This can be seen from our proof of Proposition 1. Therefore, layers in $\phi_{\mathbf{Z}}$ do not need be evaluated, and this is reflected in L4 of Algorithm B.1.

Algorithm B.1: MARGINALIZE$(c, \mathbf{y}, \mathbf{Z})$

Input: A structured-decomposable tensorized circuit c over variables $\mathbf{X}$ that is orthonormal (Definition 3); a set of variables $\mathbf{Z}$ to marginalize, and an assignment $\mathbf{y}$ to variables $\mathbf{Y} = \mathbf{X} \setminus \mathbf{Z}$.
Output: The marginal likelihood $p(\mathbf{y}) = \int_{\mathsf{dom}(\mathbf{Z})} |c(\mathbf{y}, \mathbf{z})|^2\mathrm{d}\mathbf{z}$.

1: $\mathsf{out} \leftarrow \mathsf{Map}$ ▷ A map from layers ℓ in c to their output vector.
2: $\mathsf{mar} \leftarrow \mathsf{Map}$ ▷ A map from layers ℓ to the integral $\int_{\mathsf{dom}(\mathbf{Z}')} \ell(\mathbf{y}', \mathbf{z}') \otimes \ell(\mathbf{y}', \mathbf{z}')^* \mathrm{d}\mathbf{z}'$ with $\mathbf{Z}' = \mathsf{sc}(\ell) \cap \mathbf{Z}$, $\mathbf{Y}' = \mathsf{sc}(\ell) \cap \mathbf{Y}$
3: **for** $\ell \in$ TOPOLOGICALORDERING(c) **do**
4: **if** $\mathsf{sc}(\ell) \subseteq \mathbf{Z}$ **then** SKIP ▷ Skip to the next layer
5: **if** ℓ is an input layer **then**
6: $\mathsf{out}[\ell] \leftarrow \ell(\mathbf{y}')$ ▷ $\mathbf{y}'$ denotes $\mathbf{y}$ restricted to $\mathbf{Y}'$
7: SKIP
8: **if** ℓ is a sum layer with input ℓ_1 parameterized by $\mathbf{W}$ **then**
9: **if** $\mathsf{sc}(\ell) \cap \mathbf{Z} = \varnothing$ **then** ▷ Evaluate ℓ *without* squaring
10: $\mathsf{out}[\ell] \leftarrow \mathbf{W}\mathsf{out}[\ell_1]$
11: **else** ▷ Evaluate the corresponding squared layer ℓ^2
12: $\mathsf{mar}[\ell] \leftarrow (\mathbf{W} \otimes \mathbf{W}^*)\mathsf{mar}[\ell_1]$
13: SKIP
14: ▷ ℓ is either an Hadamard or Kronecker product layer
15: $(\ell_1, \ell_2) \leftarrow$ GETINPUTS(ℓ)
16: **if** $\mathsf{sc}(\ell) \cap \mathbf{Z} = \varnothing$ **then** ▷ ℓ depends on $\mathbf{Y}$ only
17: ▷ Evaluate the product layer ℓ *without* squaring it
18: **if** ℓ is an Hadamard product layer **then**
19: $\mathsf{out}[\ell] \leftarrow \mathsf{out}[\ell_1] \odot \mathsf{out}[\ell_2]$
20: **else** ▷ ℓ is a Kronecker product layer
21: $\mathsf{out}[\ell] \leftarrow \mathsf{out}[\ell_1] \otimes \mathsf{out}[\ell_2]$
22: SKIP
23: ▷ ℓ is a product layer depending on both $\mathbf{Y}$ and $\mathbf{Z}$
24: **if** $\mathsf{sc}(\ell_1) \subseteq \mathbf{Z}$ **then** ▷ ℓ_1 depends on $\mathbf{Z}$ only
25: $\mathbf{o}_1 \leftarrow \mathsf{vec}(\mathbf{I}_{K_{\ell_1}})$ ▷ K_{ℓ_1} denotes the width of ℓ_1
26: **else if** $\mathsf{sc}(\ell_1) \cap \mathbf{Z} = \varnothing$ **then** ▷ ℓ_1 depends on $\mathbf{Y}$ only
27: $\mathbf{o}_1 \leftarrow \mathsf{out}[\ell_1] \otimes \mathsf{out}[\ell_1]^*$
28: **else** ▷ $\mathsf{sc}(\ell_1)$ depends on both $\mathbf{Y}$ and $\mathbf{Z}$
29: $\mathbf{o}_1 \leftarrow \mathsf{mar}[\ell_1]$
30: **repeat** L19-L24 by replacing ℓ_1 with ℓ_2 to obtain $\mathbf{o}_2$
31: ▷ Evaluate the corresponding squared layer ℓ^2
32: **if** ℓ is an Hadamard product layer **then**
33: $\mathsf{mar}[\ell] \leftarrow \mathbf{o}_1 \odot \mathbf{o}_2$
34: **else** ▷ ℓ is a Kronecker product layer
35: $\mathsf{mar}[\ell] \leftarrow \mathbf{P}(\mathbf{o}_1 \otimes \mathbf{o}_2)$,
36: where $\mathbf{P}$ is a permutation matrix (Appendix A)
37: **return** $\mathsf{mar}[$OUTPUTLAYER$(c)]$

Case (ii) layers depending on variables both in $\mathbf{Y}$ and $\mathbf{Z}$. Consider a layer $\boldsymbol{\ell}$ in c such that $\mathsf{sc}(\boldsymbol{\ell}) \cap \mathbf{Y} \neq \varnothing$ and $\mathsf{sc}(\boldsymbol{\ell}) \cap \mathbf{Z} \neq \varnothing$, i.e., $\boldsymbol{\ell} \in \phi_{\mathbf{Y},\mathbf{Z}}$ by hypothesis. Moreover, let $\mathbf{X}' = \mathsf{sc}(\boldsymbol{\ell}) \subseteq \mathbf{X}$, $\mathbf{Z}' = \mathbf{X}' \cap \mathbf{Z}$, and $\mathbf{Y}' = \mathbf{X}' \setminus \mathbf{Z}'$. Since input layers can only be univariate, $\boldsymbol{\ell}$ must be either a sum or product layer.

Assume $\boldsymbol{\ell}$ is a sum layer in c receiving input from $\boldsymbol{\ell}_1$ and is parameterized by $\mathbf{W} \in \mathbb{C}^{K_1 \times K_2}$. Then, the corresponding squared layer $\boldsymbol{\ell}^2$ in c^2 receives input from $\boldsymbol{\ell}_1^2$ and is parameterized by $\mathbf{W} \otimes \mathbf{W}^*$. Therefore, we have that

$$\int_{\mathsf{dom}(\mathbf{Z}')} \boldsymbol{\ell}^2(\mathbf{y}', \mathbf{z}')\mathrm{d}\mathbf{z}' = (\mathbf{W} \otimes \mathbf{W}^*) \int_{\mathsf{dom}(\mathbf{Z}')} \boldsymbol{\ell}_1^2(\mathbf{y}', \mathbf{z}')\mathrm{d}\mathbf{z}',$$

hence the integral is simply "*pushed*" towards the sub-circuit of c^2 rooted in $\boldsymbol{\ell}_1^2$. The computation of the above integral for a sum layer can be found at L12 of Algorithm B.1.

Now, assume $\boldsymbol{\ell}$ is an Hadamard product layer in c receiving input from $\boldsymbol{\ell}_1, \boldsymbol{\ell}_2$ having scopes $\mathbf{X}_1', \mathbf{X}_2'$, respectively. Then, the corresponding squared layer $\boldsymbol{\ell}^2$ in c^2 is an Hadamard layer receiving inputs from $\boldsymbol{\ell}_1$ and $\boldsymbol{\ell}_2$ (see Appendix A). Moreover, let $\mathbf{Y}_1' = \mathbf{X}_1' \cap \mathbf{Y}$, $\mathbf{Z}_1' = \mathbf{X}_1' \cap \mathbf{Z}$, $\mathbf{Y}_2' = \mathbf{X}_2' \cap \mathbf{Y}$, $\mathbf{Z}_2' = \mathbf{X}_2' \cap \mathbf{Y}$. Below we proceed by cases in order to prove L23-36 in Algorithm B.1. Due to decomposability of c (Definition 2), if $\mathbf{Z}_1' = \varnothing$ and $\mathbf{Y}_2' = \varnothing$ we recover that

$$\begin{aligned}\int_{\mathsf{dom}(\mathbf{Z}_2')} \boldsymbol{\ell}^2(\mathbf{y}_1', \mathbf{z}_2')\mathrm{d}\mathbf{z}' &= \int_{\mathsf{dom}(\mathbf{Z}_2')} \boldsymbol{\ell}^2(\mathbf{y}_1') \odot \boldsymbol{\ell}^2(\mathbf{z}_2')\mathrm{d}\mathbf{z}_2' \\ &= \boldsymbol{\ell}_1^2(\mathbf{y}_1') \odot \int_{\mathsf{dom}(\mathbf{Z}_2')} \boldsymbol{\ell}_2^2(\mathbf{z}_2')\mathrm{d}\mathbf{z}_2' \\ &= (\boldsymbol{\ell}_1(\mathbf{y}_1') \otimes \boldsymbol{\ell}_1(\mathbf{y}_1')^*) \odot \mathsf{vec}(\mathbf{I}_K),\end{aligned}$$

because $\boldsymbol{\ell}_1^2(\mathbf{y}') = \boldsymbol{\ell}_1(\mathbf{y}') \odot \boldsymbol{\ell}_1(\mathbf{y}')^*$ and $\boldsymbol{\ell}_2$ depends on $\mathbf{Z}$ only (see **Case (i)**) above. Therefore, for this case we do not need to square the sub-circuit rooted in $\boldsymbol{\ell}_1$, i.e., we can evaluate $\boldsymbol{\ell}_1$ as is on the input $\mathbf{y}_1'$ and then computes the Kronecker product of the output vector with its conjugate only. Conversely, if $\mathbf{Y}_1' = \varnothing$ and $\mathbf{Z}_2' = \varnothing$ we recover a similar result: we do not need to square the sub-circuits rooted in $\boldsymbol{\ell}_2$. This case is captured by L24-27 in Algorithm B.1.

Furthermore, consider the case $\mathbf{Z}_1' = \varnothing$, then similarly to the above we recover that

$$\begin{aligned}&\int_{\mathsf{dom}(\mathbf{Z}_2')} \boldsymbol{\ell}^2(\mathbf{y}_1', \mathbf{z}_2')\mathrm{d}\mathbf{z}' \\ &= (\boldsymbol{\ell}_1(\mathbf{y}_1') \otimes \boldsymbol{\ell}_1(\mathbf{y}_1')^*) \odot \int_{\mathsf{dom}(\mathbf{Z}_2')} \boldsymbol{\ell}_2^2(\mathbf{y}_2', \mathbf{z}_2')\mathrm{d}\mathbf{z}_2'.\end{aligned}$$

Therefore, while we do not require squaring the sub-circuit rooted in $\boldsymbol{\ell}_1$, we however need to square the one rooted in $\boldsymbol{\ell}_2$ and integrate it. Conversely, we recover a similar result if $\mathbf{Z}_2' = \varnothing$. This case is captured by L28-29 in Algorithm B.1.

If neither $\mathbf{Z}_1'$ nor $\mathbf{Z}_2'$ are empty, then we need to square the sub-circuits rooted both in $\boldsymbol{\ell}_1$ and $\boldsymbol{\ell}_2$, since we have that

$$\begin{aligned}&\int_{\mathsf{dom}(\mathbf{Z}_1' \cup \mathbf{Z}_2')} \boldsymbol{\ell}^2(\mathbf{y}_1'\mathbf{y}, \mathbf{z}_2')\mathrm{d}\mathbf{z}' \\ &= \int_{\mathsf{dom}(\mathbf{Z}_1')} \boldsymbol{\ell}_1^2(\mathbf{y}_1', \mathbf{z}_1')\mathrm{d}\mathbf{z}_1' \odot \int_{\mathsf{dom}(\mathbf{Z}_2')} \boldsymbol{\ell}_2^2(\mathbf{y}_1', \mathbf{z}_2')\mathrm{d}\mathbf{z}_2'.\end{aligned}$$

Instead of Hadamard product layers, a similar discussion can be carried for the case of $\boldsymbol{\ell}$ being a Kronecker product layer. In particular, the computation of the above integrals for Hadamard or Kronecker product layers for the discussed cases can be found at L23-36 of Algorithm B.1.

We now discuss what is the computational complexity for the **Case (ii)** above. First, we observe that we need to quadratically increase the size of each layer that depends on both variables in $\mathbf{Y}$ and in $\mathbf{Z}$. This already requires time $\mathcal{O}(|\phi_{\mathbf{Y},\mathbf{Z}}|S^2)$. In addition, we need to compute a Kronecker product of the outputs of layers that depend on variables $\mathbf{Y}$ only, but that are also input to product layers depending on both $\mathbf{Y}$ and $\mathbf{Z}$. However, since (i) each product layer depending on both $\mathbf{Y}$ and $\mathbf{Z}$ receives input from exactly two other layers $\boldsymbol{\ell}_1, \boldsymbol{\ell}_2$, and (ii) at most one between $\boldsymbol{\ell}_1$ and $\boldsymbol{\ell}_2$ can depend on variables $\mathbf{Y}$ only, we recover that the complexity of computing these Kronecker products is $\mathcal{O}(|\phi_{\mathbf{Y},\mathbf{Z}}|J^2)$, where J is the maximum output size of each layer in c. Since the output size of each layer J is always bounded by the layer size S (see below Definition 1), it turns out that $|\phi_{\mathbf{Y},\mathbf{Z}}|J^2 \in \mathcal{O}(|\phi_{\mathbf{Y},\mathbf{Z}}|S^2)$. Therefore, the Kronecker products mentioned above account for only a constant multiplicative factor in our complexity.

Case (iii) layers depending on variables $\mathbf{Y}$ only. In **Case (ii)** we have shown that we need to evaluate the layers in c whose scope depends on variable $\mathbf{Y}$ only. Since such layers do not need to be squared (see above), it turns out computing them requires time $\mathcal{O}(|\phi_{\mathbf{Y}}|S)$. This case is captured by L9-10 and L16-L22 in Algorithm B.1.

Therefore, by combining **Cases (i-iii) above**, we recover the overall time complexity of Algorithm B.1 is $\mathcal{O}(|\phi_{\mathbf{Y}}|S + |\phi_{\mathbf{Y},\mathbf{Z}}|S^2)$. □

B.3 Are Orthonormal Circuits less Expressive?

Theorem 2. Let c be a tensorized circuit over variables $\mathbf{X}$. Assume that each input layer in c encodes a set of orthonormal functions. Then, there exists an algorithm returning an orthonormal circuit c' in polynomial time such that c' is equivalent to c up to a multiplicative constant, i.e., $c'(\mathbf{X}) = Z^{-\frac{1}{2}} c(\mathbf{X})$ where $Z = \int_{\mathsf{dom}(\mathbf{X})} |c(\mathbf{x})|^2 \mathrm{d}\mathbf{x}$.

Proof. For the proof, we will show the correctness of our Algorithm 1 as to retrieve a tensorized orthonormal circuit c' from c such that $c'(\mathbf{X}) = \beta c(\mathbf{X})$ for a positive real constant β. Before showing this, we observe that, since c' is orthonormal, then

$$p(\mathbf{X}) = |c'(\mathbf{X})|^2 = \beta^2 |c(\mathbf{X})|^2.$$

Therefore, we must have that $\beta = Z^{-\frac{1}{2}}$ with Z being the partition function of c^2, i.e., $Z = \int_{\mathsf{dom}(\mathbf{X})} |c(\mathbf{x})|^2 \mathrm{d}\mathbf{x}$. In other words, it turns out Algorithm 1 not only returns c', but also the value $\beta = Z^{-\frac{1}{2}}$ thus implicitly computing the partition function. It remains to show the correctness of Algorithm 1 as mentioned above.

Let c be a structured-decomposable tensorized circuit whose input layers encode sets of orthonormal functions. We show by structural induction how the orthonormal circuit c'

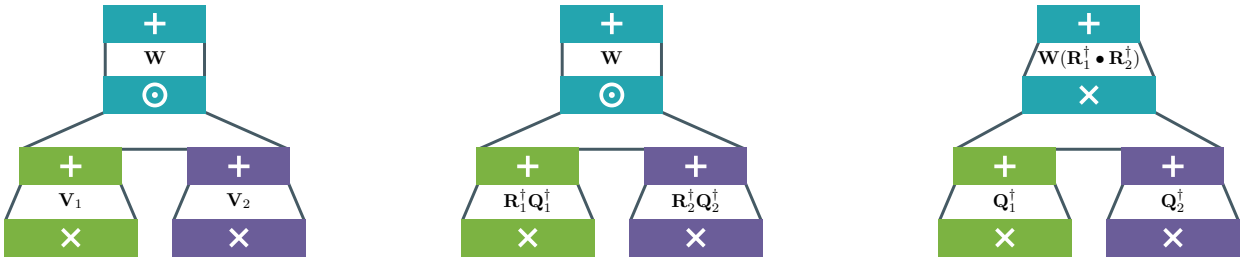

Figure B.1: **Algorithm 1 recursively make the sum layer parameter matrices of (semi-)unitary.** Given *a fragment* of a tensorized circuit (left), our algorithm computes QR decompositions of the sum layer parameter matrices $\mathbf{V}_1^\dagger$ and $\mathbf{V}_2^\dagger$, thus yielding $\mathbf{V}_1 = \mathbf{R}_1^\dagger\mathbf{Q}_1^\dagger$ and $\mathbf{V}_2 = \mathbf{R}_2^\dagger\mathbf{Q}_2^\dagger$ (mid) (L9-13 in the algorithm). The matrices $\mathbf{R}_1^\dagger, \mathbf{R}_2^\dagger$ are propagated towards the subsequent Hadamard layer in Algorithm 1, where $\mathbf{R}_1^\dagger \bullet \mathbf{R}_2^\dagger$ is computed (L21) and then multiplied to the parameter matrix $\mathbf{W}$ (right) (L8). Note that the Hadamard product layer is replaced with a Kronecker product layer, accounting for a polynomial increase in the layer size. The same procedure is then recursively applied to the parameter matrix $\mathbf{W}(\mathbf{R}_1^\dagger \bullet \mathbf{R}_2^\dagger)$ (not shown).

is constructed from c using Algorithm 1. The idea is to apply QR decompositions to the sum layer parameters, retain the (semi-)unitary matrix of the decomposition and "*push*" the upper-triangular matrix towards the output layer of the circuit in a bottom-up fashion. For this reason, after each recursive step Algorithm 1 also returns a matrix $\mathbf{R}$, which can be a wide matrix and in general it is not (semi-)unitary. In particular, given $(\ell', \mathbf{R})$ the output of Algorithm 1 for a layer ℓ, we associate the semantics $\ell(\mathsf{sc}(\ell)) = \mathbf{R}\ell'(\mathsf{sc}(\ell))$ to it, where the circuit rooted in ℓ' is orthonormal by inductive hypothesis. Note that for any input layer ℓ in c encoding K orthonormal functions, we assume ℓ is also in c' and $\mathbf{R} = \mathbf{I}_K$ (see L3 of the algorithm).

Case (i): sum layer. Let ℓ be a sum layer with scope $\mathsf{sc}(\ell) = \mathbf{Z}$ and computing the matrix-vector product $\mathbf{W}\ell_1(\mathbf{Z})$, with $\mathbf{W} \in \mathbb{C}^{K_1 \times K_2}$, $K_1 \leq K_2$. By applying Algorithm 1 on the circuit rooted in ℓ_1, we retrieve the layer ℓ_1' and the matrix $\mathbf{R}_1 \in \mathbb{C}^{K_2 \times K_3}$ such that $\ell_1(\mathbf{Z}) = \mathbf{R}_1\ell_1'(\mathbf{Z})$ holds. Therefore, we can write the function computed by ℓ as $\ell(\mathbf{Z}) = \mathbf{W}\mathbf{R}_1\ell_1'(\mathbf{Z})$. Let $\mathbf{V} = \mathbf{W}\mathbf{R}_1 \in \mathbb{C}^{K_1 \times K_3}$, with $K_1 \leq K_3$. To retrieve a sum layer parameterized by a (semi-)unitary matrix, we apply the QR decomposition on $\mathbf{V}^\dagger$, i.e., $\mathbf{V}^\dagger = \mathbf{Q}\mathbf{R}$, where $\mathbf{Q} \in \mathbb{C}^{K_3 \times K_1}$ and $\mathbf{R} \in \mathbb{C}^{K_1 \times K_1}$. Here, $\mathbf{Q}^\dagger\mathbf{Q} = \mathbf{I}_{K_1}$ and $\mathbf{R}$ is an upper triangular matrix. Using the QR decomposition above, we can rewrite $\ell(\mathbf{Z}) = \mathbf{R}^\dagger\mathbf{Q}^\dagger\ell_1'(\mathbf{Z})$. Finally, we retrieve a sum layer ℓ' in c' computing $\ell'(\mathbf{Z}) = \mathbf{Q}^\dagger\ell_1'(\mathbf{Z})$, i.e., $\ell(\mathbf{Z}) = \mathbf{R}^\dagger\ell'(\mathbf{Z})$. Thus, L11 in Algorithm 1 returns both ℓ' and $\mathbf{R}^\dagger$.

Case (ii): Kronecker product layer. Consider the case ℓ is an Kronecker product layer having scope $\mathbf{Z} = \mathsf{sc}(\ell) = \mathsf{sc}(\ell_1) \cup \mathsf{sc}(\ell_2)$, and computing $\ell(\mathbf{Z}) = \ell_1(\mathbf{Z}_1) \otimes \ell_2(\mathbf{Z}_2)$. By inductive hypothesis, let ℓ_1' and ℓ_2' be the output layers of orthonormal circuits obtained by applying Algorithm 1 on ℓ_1 and ℓ_2, respectively. Moreover, let $\mathbf{R}_1 \in \mathbb{C}^{K_1 \times K_2}$ and $\mathbf{R}_2 \in \mathbb{C}^{K_3 \times K_4}$, with $K_1 \leq K_2$ and $K_3 \leq K_4$, be the wide matrices returned by the algorithm. Therefore, we have that ℓ_1 (resp. ℓ_2) computes $\ell_1(\mathbf{Z}_1) = \mathbf{R}_1\ell_1'(\mathbf{Z}_1)$ (resp. $\ell_1(\mathbf{Z}_2) = \mathbf{R}_2\ell_2'(\mathbf{Z}_2)$). For this reason, we can rewrite the function computed by ℓ as

$$\begin{aligned}\ell(\mathbf{Z}) &= (\mathbf{R}_1\ell_1'(\mathbf{Z}_1)) \otimes (\mathbf{R}_2\ell_2'(\mathbf{Z}_2)) \\ &= (\mathbf{R}_1 \otimes \mathbf{R}_2)(\ell_1'(\mathbf{Z}_1) \otimes \ell_2'(\mathbf{Z}_2)),\end{aligned}$$

where we use the Kronecker mixed-product property. Finally, we retrieve a Kronecker layer ℓ' in c' computing $\ell'(\mathbf{Z}) = \ell_1'(\mathbf{Z}_1) \otimes \ell_2'(\mathbf{Z}_2)$, i.e., $\ell(\mathbf{Z}) = (\mathbf{R}_1 \otimes \mathbf{R}_2)\ell'(\mathbf{Z})$. Thus, L16 in Algorithm 1 returns both ℓ' and $\mathbf{R}_1 \otimes \mathbf{R}_2$.

Case (iii): Hadamard product layer. Similarly, we consider the case of ℓ being an Hadamard product layer having scope $\mathbf{Z} = \mathsf{sc}(\ell) = \mathsf{sc}(\ell_1) \cup \mathsf{sc}(\ell_2)$, and computing $\ell(\mathbf{Z}) = \ell_1(\mathbf{Z}_1) \otimes \ell_2(\mathbf{Z}_2)$. By inductive hypothesis, let ℓ_1' and ℓ_2' be the output layers of orthonormal circuits obtained by applying Algorithm 1 on ℓ_1 and ℓ_2, respectively. Moreover, let $\mathbf{R}_1 \in \mathbb{C}^{K_1 \times K_2}$ and $\mathbf{R}_2 \in \mathbb{C}^{K_1 \times K_3}$, with $K_1 \leq K_2$ and $K_1 \leq K_3$, be the wide matrices returned by the algorithm. Therefore, we have that ℓ_1 (resp. ℓ_2) computes $\ell_1(\mathbf{Z}_1) = \mathbf{R}_1\ell_1'(\mathbf{Z}_1)$ (resp. $\ell_1(\mathbf{Z}_2) = \mathbf{R}_2\ell_2'(\mathbf{Z}_2)$). For this reason, we can rewrite the function computed by ℓ as

$$\begin{aligned}\ell(\mathbf{Z}) &= (\mathbf{R}_1\ell_1'(\mathbf{Z}_1)) \odot (\mathbf{R}_2\ell_2'(\mathbf{Z}_2)) \\ &= (\mathbf{R}_1 \bullet \mathbf{R}_2)(\ell_1'(\mathbf{Z}_1) \otimes \ell_2'(\mathbf{Z}_2)),\end{aligned}$$

where for the last equality we used the Hadamard mixed-product property, and $\bullet$ denotes the face-splitting matrix product operator as defined next.

Definition B.1 (Face-splitting product)**.** Let $\mathbf{A} \in \mathbb{C}^{m \times k}$ and $\mathbf{B} \in \mathbb{C}^{m \times r}$ be matrices. The face-splitting product $\mathbf{A} \bullet \mathbf{B}$ is defined as the matrix $\mathbf{C} \in \mathbb{C}^{m \times kr}$,

$$\mathbf{C} = \begin{bmatrix} \mathbf{a}_1 \otimes \mathbf{b}_1 \\ \vdots \\ \mathbf{a}_m \otimes \mathbf{b}_m \end{bmatrix} \quad \text{where} \quad \mathbf{A} = \begin{bmatrix} \mathbf{a}_1 \\ \vdots \\ \mathbf{a}_m \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_m \end{bmatrix},$$

and $\{\mathbf{a}_i\}_{i=1}^m, \{\mathbf{b}_i\}_{i=1}^m$ are row vectors.

Finally, we retrieve a Kronecker layer ℓ' in c' computing $\ell'(\mathbf{Z}) = \ell_1'(\mathbf{Z}_1) \otimes \ell_2'(\mathbf{Z}_2)$, i.e., $\ell(\mathbf{Z}) = (\mathbf{R}_1 \bullet \mathbf{R}_2)\ell'(\mathbf{Z})$. Thus, L21 in Algorithm 1 returns both ℓ' and $\mathbf{R}_1 \bullet \mathbf{R}_2$. We note that the Hadamard layer is replaced by a Kronecker

layer (see e.g. Fig. B.1), thus **Case (iii)** is the only case accounting for a polynomial increase in the size of the layer.

Case (iv): output layer. Finally, we consider the case of ℓ being the output layer in c, thus resulting in the last step of the recursion. W.l.o.g. we consider ℓ being a sum layer. Then from our **Case (i)** above, we have that $\mathbf{R} \in \mathbb{C}^{1\times 1}$ is obtained by the QR decomposition of a column vector $\mathbf{V}^\dagger \in \mathbb{C}^{K\times 1}$, thus corresponding to the scalar r_{11} such that $||r_{11}\mathbf{V}^\dagger||_2 = 1$, i.e., $r_{11} = ||\mathbf{V}^\dagger||_2^{-1} = \left(\sum_{i=1}^K |v_{i1}|^2\right)^{-\frac{1}{2}}$. Therefore, the non-negative scalar β mentioned at the beginning of our proof must be $\beta = r_{11} = Z^{-\frac{1}{2}}$. Hence, $Z = \sum_{i=1}^K |v_{i1}|^2$.

Finally, the computational complexity of Algorithm 1 mainly depends on the complexity of performing QR decompositions and computing Kronecker products of matrices. In particular, we need to perform as many QR decompositions as the number of sum layers in c, each requiring time $\mathcal{O}(K_2^3)$ in the case of a matrix $\mathbf{V} \in \mathbb{C}^{K_1\times K_3}$, $K_1 \leq K_3$, is $\mathcal{O}(K_3^3)$. In addition, in the worst case c consists of only Kroneckers as product layers, we have to compute Kronecker products between matrices $\mathbf{R}_1 \in \mathbb{C}^{K_1\times K_2}$ and $\mathbf{R}_2 \in \mathbb{C}^{K_3\times K_4}$, each requiring time $\mathcal{O}(K_1K_2K_3K_4)$. Assuming matrix products require asymptotic cubic time, we recover the overall complexity of Algorithm 1 is $\mathcal{O}(L_{\mathsf{sum}}J^3 + L_{\mathsf{prod}}J^4)$, where J is the maximum output size of each layer in c and L_{sum} (resp. L_{prod}) is the number of sum (resp. product) layers in c. □